Project: Hangman

We're going to write the game of Hangman. This document provides a step-by-step approach to help you build the game. Use it as much or as little as you want. If you're uncertain, I recommend sticking with the document; however, if you want to try attacking this program on your own, that's great too. Actual coding starts in question 2.

You only have to pass in the Hangman code. You *don't* have to pass in answers to all the intermediate questions.

1. Remember the maximum value trick we covered yesterday? Here's another problem along the same vain:

Let's say I want to check and see if a number of facts are ALL true. For example, is **every element in a list** less than 6?

Use what you learned from yesterday (and the homework) to write a *for* loop that will determine if all elements in a variable *some_list* are less than 6. Check your code using *some_list = [1,5,3,4]* and on *some_list = [5,3,7,5]*.

We can also test if AT LEAST ONE fact is true. Write a *for* loop that will determine if **at least one** element in a list is less than 6. Test on [7,8,7,9] and [7,2,5,8]

The first is the equivalent of checking *A and B and C and D and ...* The second is the equivalent of checking *A or B or C or D or ...*

Throw out this code when you've finished the problem – we just wanted to make sure you can solve problems of this type (you'll need it for Hangman.)

2. Create a new file *hangman.py*. We're going to start by storing the **state** of the game in variables at the top of the program. The state is a complete description of all the information about the game. In Nim, the state would be:

- The current player
- How many stones are in the pile

For Hangman, we need to store 3 pieces of information:

secret_word: The word they are trying to guess (string).
letters_guessed: The letters that they have guessed so far (list).
mistakes_made: The number of incorrect guesses they've made so far (int).

You can name these something else if you'd like, but **use a descriptive name**.

For now, set *secret_word* to be "claptrap." Once we've finished our program and got it working, then we'll add a prompt at the beginning of the program to let a friend of the user choose the word. (This is called incremental programming – instead of trying to get everything right the first time, we'll get the basic program working then incrementally add small portions of code.) "claptrap" was selected because it's reasonably long and has duplicate letters -- hopefully that will allow us to catch any bugs we might make.

Question -- why can't we use *len(letters_guessed)* for *mistakes_made*?

1b. Also create a constant variable at the top:

Max_guesses = 6.

Constant just means that we won't change it. This isn't enforced by the compiler, so be careful not to accidentally change the value of *Max_guesses*. My style is to begin variables that I don't plan to change with a capital letter -- other people do different things (some would have written MAX_GUESSES, for example.) Any way works.

We can decide what to do with this at the end (should we have an "easy", "medium", "hard" mode with different guesses?)

Idea: At the end of the program, we should change “claptrap” to something with more than 6 distinct letters to make sure that the program doesn't accidentally increment the number of mistakes on a correct guess.

Pre-3. Quickie reminder: Enter the following lines of code in the prompt

```
for i in "hello":
  print i

for i in ['a',True,123]:
  print i
```

Just a reminder on how *for* loops work.

3. Let's start writing code! Here’s our approach..we’ll write functions to take care of smaller tasks that we need to do in hangman, then use them to write the actual game itself.

First, write the function *word_guessed()*. *word_guessed()* will return True if the player has successfully guessed the word, and False otherwise.

Example:
If the *letters_guessed* variable has the value *['a','l','m','c','e','t','r','p','n']*, *word_guessed()* will return True. If the *letters_guessed* variable has the value *['e','l','q','t','r','p','n']*, *word_guessed()* will return False.

Hint: Obviously, you'll use a loop. There are two things you could loop over -- the letters in *secret_word* or the letters in *letters_guessed*. Which one do you want to loop over? Don't just guess here, think! One of them makes sense / will be a lot easier than the other. You'll also be using the trick from the first problem.

4. Try this: type the following commands into the prompt.

```
dir()
a = 5
dir()
b = 3
c = 7
a = 14
dir()
from string import *
dir()
```

What does the *dir* function do?

While still at the prompt, type *help(join)* and *help(lower)*.

4b. What lines of code belong in the missing spaces to achieve the desired outcome?

```
List1 = ['H','e','a','r']
missing
missing
print string1 #prints 'hear'
```

5. Back to Hangman. So you'll want to use the string library. Add *from string import ** to the top of your program.

6. Now write a function *print_guessed()* (or whatever you want to name it) that returns a string that contains the word with a dash '-' in place of letters not guessed yet.

Example:
If the *letters_guessed* variable *has the value ['a','p']*, the expression *print print_guessed()* will print `--ap--ap`.
If the *letters_guessed* variable *has the value []*, the expression *print print_guessed()* will print `--------`.
If the *letters_guessed* variable *has the value ['a','l','m','c','e','t','r','p','n']*, the expression *print print_guessed()* will print `claptrap`.

Hint: There are a lot of ways to go about this. One way is to iterate through *secret_word* and append the character you want to print to a list. Then use the *join* function to change the list into a string: your last line will look something like *return join(character_list, "")*

7. Now write the main game code. It may help to informally sketch out the code you want to write, e.g
continually loop {
print n guesses left
print word
get letter in lowercase
has letter already been guessed?
is letter in word?
If so, what should I do? If not, what should I do?}

(remember the **break** statement if you use the continual loop)

8. Congratulations! You've finished the game. Now we want to make it look pretty so everyone else will be impressed as we are :p. Polish your game a bit (don't just use the word claptrap every time)

Hint: If you put *from random import ** at the top of your code, you can use the *randint(a,b)* function – it returns a random number between *a* and *b* (inclusive.) This is optional, though (you could just prompt them too.)